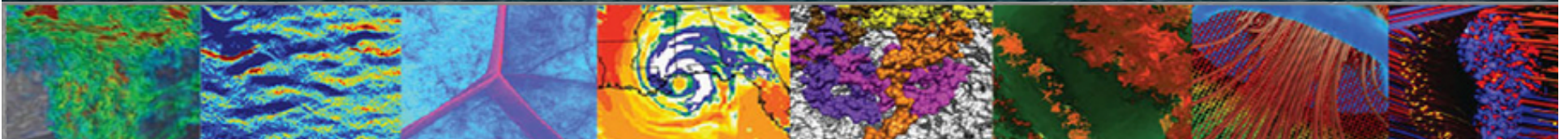



Algorithmic Adaptations to Extreme Scale

David Keyes

**Division of Computer, Electrical and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology**



The image shows two hands against a dramatic, cloudy sky. The hand on the left is open, with fingers slightly curled, reaching towards a red baton. The hand on the right is firmly gripping the baton. The baton is a solid, bright red cylinder. The text 'Energy-aware generation' is overlaid on the left hand, and 'BSP generation' is overlaid on the right hand.

Energy-aware
generation

BSP
generation

To paraphrase Benjamin Franklin:

“An [infrastructure], if you can keep it.”

A conclusion up front:

- Hope for *some* of today's best algorithms to make the leap
 - ◆ greater concurrency
 - ◆ less synchrony
 - ◆ built-in resilience (“algorithm-based fault tolerance” or ABFT)
- Programming models will have to be severely stretched
- Everything should be “on the table” for trades “over the threshold”

Motivation

- **High performance with high productivity on “the multis”:**
 - ◆ **Multi-scale, multi-physics problems in multi-dimensions**
 - ◆ **Using multi-models and/or multi-levels of refinement**
 - ◆ **Exploiting polyalgorithms in multiple precisions in multi-protocol programming styles**
 - ◆ **On multi-core, massively multi-processor systems**
 - ◆ **Requiring a multi-disciplinary approach**

Motivation

- **High performance with high(-est possible) productivity on “the multis”:**
 - ◆ **Multi-scale, multi-physics problems in multi-dimensions**
 - ◆ **Using multi-models and/or multi-levels of refinement**
 - ◆ **Exploiting polyalgorithms in multiple precisions in multi-protocol programming styles**
 - ◆ **On multi-core, massively multi-processor systems**
 - ◆ **Requiring a multi-disciplinary approach**

Can't cover all this in one talk...

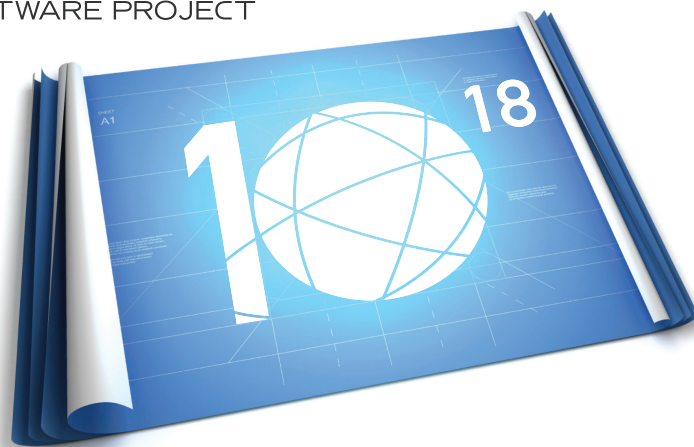
Given the architectural stresses, how can new algorithms help?

Purpose of the presentation

- Increase quality of “co-design” dialog between application user-developers and systems software and hardware developers
- Vendors are surprisingly willing
 - ◆ No longer one type of vendor
 - ◆ All vendors motivated by *some* mass market
 - Low-power (smartphones, remote sensors, etc.)
 - High graphics throughput (gaming, entertainment, etc.)
 - High reliability (business, data centers, etc.)
 - ◆ Unfortunately, computational scientists want all three
 - ... and we are *a relatively* small market

For background, see
www.exascale.org/iesp

INTERNATIONAL
EXASCALE ROADMAP 1.0
SOFTWARE PROJECT



Jack Dongarra
Pete Beckman
Terry Moore
Patrick Aerts
Giovanni Aloisio
Jean-Claude Andre
David Barkai
Jean-Yves Berthou
Taisuke Boku
Bertrand Braunschweig
Franck Cappello
Barbara Chapman
Xuebin Chi

Alok Choudhary
Sudip Dosanjh
Thom Dunning
Sandro Fiore
Al Geist
Bill Gropp
Robert Harrison
Mark Hereld
Michael Heroux
Adolfo Hoisie
Koh Hotta
Yutaka Ishikawa
Fred Johnson

Sanjay Kale
Richard Kenway
David Keyes
Bill Kramer
Jesus Labarta
Alain Lichnewsky
Thomas Lippert
Bob Lucas
Barney Maccabe
Satoshi Matsuoka
Paul Messina
Peter Michielse
Bernd Mohr

Matthias Mueller
Wolfgang Nagel
Hiroshi Nakashima
Michael E. Papka
Dan Reed
Mitsuhiro Sato
Ed Seidel
John Shalf
David Skinner
Marc Snir
Thomas Sterling
Rick Stevens
Fred Streitz

Bob Sugar
Shinji Sumimoto
William Tang
John Taylor
Rajeev Thakur
Anne Trefethen
Mateo Valero
Aad van der Steen
Jeffrey Vetter
Peg Williams
Robert Wisniewski
Kathy Yelick

The International Exascale
Software Roadmap,
J. Dongarra, P. Beckman, et al.,
*International Journal of High
Performance Computer
Applications* **25**(1), 2011, ISSN
1094-3420.

SPONSORS



Extrapolating exponentials eventually fails

- Scientific computing at a crossroads w.r.t. extreme scale
- Proceeded steadily for decades from giga- (1988) to tera- (1998) to peta- (2008) with
 - ◆ *same* SPMD programming model
 - ◆ *same* assumptions about who is responsible for what
 - ◆ *same* classes of algorithms (cf. 25 yrs. of Gordon Bell Prizes)
- Exa- is qualitatively different and will be much harder
- Core numerical analysis and scientific computing will confront exascale to maintain sponsor relevance
 - ◆ not a “distraction,” but an intellectual stimulus
 - ◆ potentially big gains in adapting to new hardware environment
 - ◆ the journey will be as fun as the destination

Relevance of exascale to users today

- **Modelers are on the front line**
 - ◆ without concurrent research in the form of new models and mathematics, the passage to exascale hardware will yield little new scientific fruit
- **Scientists will find the computational power to do things many have wanted**
 - ◆ more room for creativity in “post-forward” problems (inverse problems, control, data assimilation, uncertainty quantification)
 - ◆ scientists will participate in cross-disciplinary integration – “third paradigm” *and* “fourth paradigm”
 - ◆ remember that *exascale at the lab* means *petascale on the desk*
- **We suggest some mathematical opportunities, after (quickly) reviewing the hardware challenges**

Why exa- is different

Which steps of FMADD take more energy?

64-bit floating-point fused multiply add

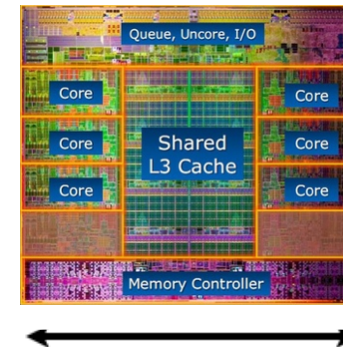
or

moving four 64-bit operands 20 mm across the die

```

      934,569.299814557      input
x      52.827419489135904    input
-----
= 49,370,884.442971624253823
+      4.20349729193958      input
-----
= 49,370,888.64646892        output

```



20 mm
(Intel Sandy Bridge, 2.27B transistors)

Going across the die will require an order of magnitude more!

DARPA study predicts that by 2019:

- ◆ Double precision FMADD flop: 11pJ
- ◆ cross-die per word access (1.2pJ/mm): 24pJ (= 96pJ overall)

after DARPA report of P. Kogge (ND) *et al.* and T. Schulthess (ETH)

ATPESC 2013

Today's power costs per operation

Operation	approximate energy cost
DP FMADD flop	100 pJ
DP DRAM read-to-register	4800 pJ
DP word transmit-to-neighbor	7500 pJ
DP word transmit-across-system	9000 pJ

Remember that a *pico* (10^{-12}) of something done *exa* (10^{18}) times per second is a *mega* (10^6)-somethings per second

- ◆ 100 pJ at 1 Eflop/s is 100 MW (for the flop/s only!)
- ◆ In the USA, 1 MW-year costs \$1M ($\$0.12/\text{KW-hr} \times 8760 \text{ hr/yr}$)
 - ◆ You “use” 1.4 KW continuously on average

Why exa- is different

**Moore's Law (1965) does not end but
Dennard's MOSFET scaling (1972) does**

Table 1
Scaling Results for Circuit Performance

Device or Circuit Parameter	Scaling Factor
Device dimension t_{ox} , L , W	$1/\kappa$
Doping concentration N_a	κ
Voltage V	$1/\kappa$
Current I	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit VC/I	$1/\kappa$
Power dissipation/circuit VI	$1/\kappa^2$
Power density VI/A	1

Table 2
Scaling Results for Interconnection Lines

Parameter	Scaling Factor
Line resistance, $R_L = \rho L/Wt$	κ
Normalized voltage drop IR_L/V	κ
Line response time $R_L C$	1
Line current density I/A	κ



Robert Dennard, IBM
(inventor of DRAM, 1966)

Eventually processing will be
limited by transmission

What will first “general purpose” exaflop/s machines look like?

- *Hardware*: many potentially exciting paths beyond today’s CMOS silicon-etched logic, but not commercially at scale within the decade
- *Software*: many ideas for general-purpose and domain-specific programming models beyond “MPI + X”, but not penetrating the main CS&E workforce within the decade
 - ◆ “X” is OpenMP, CUDA, OpenACC, pthreads, etc.

Tianhe-2 by Inspur / NUDT / Intel (June 2013)



Main system: 32K Ivy Bridge + 48K Xeon Phi chips ~55 PF/s

*Front-end: 4K Galaxy FT-1500 chips, 59 TF/s

Tianhe-2 vs. envisioned exascale hardware: a heterogeneous, distributed memory *GigaHz KiloCore MegaNode* system

	Tianhe-2 (2013)	Exa (2020)	Ratio to go
Number of nodes	16,000 (each 2 Ivy + 3 Phi)	1,000,000	~60
Node concurrency	24 Ivy + 171 Phi = 195 cores	1,000	~5
Node memory (GB)	88 Ivy + 8 Phi = 96	64	(1)
Node peak perf (GF/s)	422 Ivy + 3,009 Phi = 3,431	1,000	(1)
Total concurrency	3,120,000	1 B	~320
Total memory (PB)	1.536	64	~40
Total peak perf (PF/s)	54.9	1,000	~20
Power (MW)	17.8 (+ 24 MW cooling !)	20	(1)

after P. Beckman (ANL) *et al.*

ATPESC 2013

Some exascale themes

- **Clock rates cease to increase while arithmetic capacity continues to increase dramatically w/concurrency consistent with Moore's Law**
- **Storage capacity diverges exponentially below arithmetic capacity**
- **Transmission capacity (memory BW and network BW) diverges exponentially below arithmetic capacity**
- **Mean time between hardware interrupts shortens**
- **Billions of dollars of scientific software hang in the balance until better algorithms arrive to span the architectural gap**

Implications of operating on the edge

- **Draconian reduction required in power per flop and per byte will make computing and copying data less reliable**
 - ◆ voltage difference between “0” and “1” will be reduced
 - ◆ circuit elements will be smaller and subject to greater physical noise per signal
 - ◆ there will be more errors that must be caught and corrected
- **Power may be cycled off and on or clocks slowed and speeded based on compute schedules and based on cooling capacity**
 - ◆ makes per node performance rate unreliable

Implications of operating on the edge

- Expanding the number of nodes (processor-memory units) beyond 10^6 would *not* a serious threat to algorithms that lend themselves to well-amortized precise load balancing
 - ◆ provided that the nodes are performance reliable
- A real challenge is *usefully* expanding the number of cores on a node to 10^3
 - ◆ must be done while memory and memory bandwidth per node expand by (at best) ten-fold less (basically “strong” scaling)
- It is already orders of magnitude slower to retrieve an operand from main DRAM memory than to perform an arithmetic operation – will get worse by another
 - ◆ “almost all” operands must come from registers or upper cache

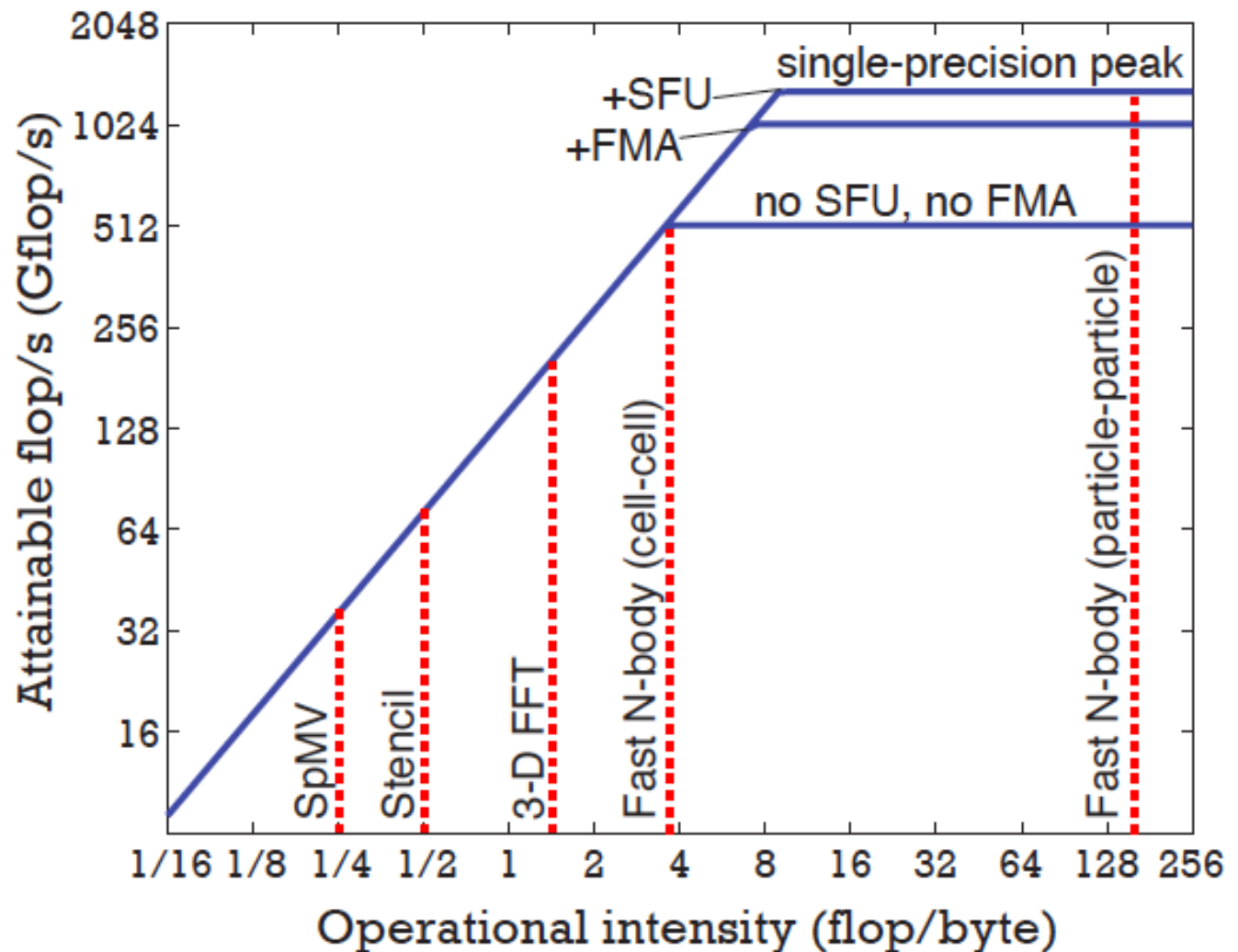
“Missing” mathematics

- **New formulations with**
 - ◆ **greater arithmetic intensity (flops per bytes moved into and out of registers and upper cache)**
 - ◆ **reduced communication**
 - ◆ **reduced synchronization**
 - ◆ **assured accuracy with (adaptively) less floating-point precision**
 - ◆ **algorithmic resilience to many types of faults**
- **Quantification of trades between limiting resources**
- ***Plus* all of the exciting analytical agendas that exascale is meant to exploit**

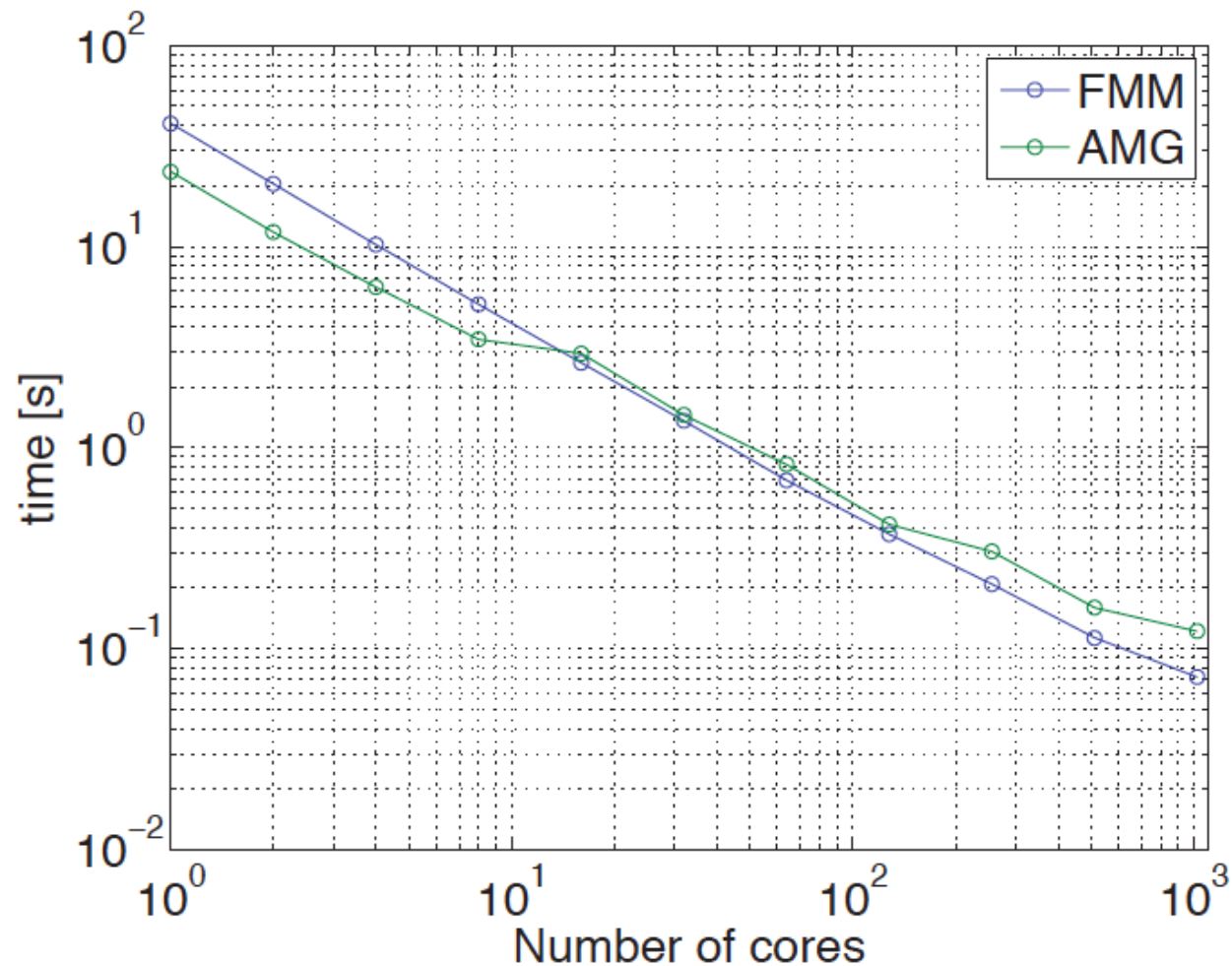
Arithmetic intensity illustration

Roofline model of numerical kernels on an NVIDIA C2050 GPU (Fermi). The ‘SFU’ label is used to indicate the use of special function units and ‘FMA’ indicates the use of fused multiply-add instructions.

(The order of fast multipole method expansions was set to $p = 15$.)



Research in progress: FMM vs AMG preconditioning, strong scaling on Stampede*



* Poisson problem, Dirichlet BCs handled via BIE for FMM (cost included)

Reduction of frequency of communication and synchronization for $Ax=b$

- **Classical: amortize communication over many power/reduce steps**
 - ◆ **s-step Krylov methods: power kernels with wide halos and extra orthogonalization**
 - ◆ **Block Krylov methods: solve b several independent systems at once with improved convergence (based on λ_{\max}/λ_b rather than $\lambda_{\max}/\lambda_{\min}$)**
 - ◆ **“tall skinny QR” ($n \times m$): recursively double the row-scope of independent QRs – $\log p$ messages for p processors (vs. $n \log p$)**
- **Invade classical steps:**
 - ◆ **operations dynamically scheduled with DAGs**
 - ◆ **NUMA-aware (local) work-stealing**

Reduction of domain of synchronization

- **Nonlinear Schwarz replaces a Newton method for a global nonlinear system, $F(u)=0$,**
 - which computes a global distributed Jacobian matrix and synchronizes globally in both the Newton step and in solving the global linear system for the Newton
- **... with a set of local problems on subsets of the global nonlinear system**
 - each local problem has only local synchronization
 - all of the linear systems for local Newton updates have only local synchronization
 - there is still global synchronization in a number of steps hopefully *much fewer* than required in the original Newton method

How are most workhorse simulations implemented at the infra-petascale today?

- **Iterative methods based on data decomposition and message-passing**
 - ◆ each individual processor works on a portion of the original problem and exchanges information at its boundaries with other processors that own portions with which it interacts causally, to evolve in time or to establish equilibrium
 - ◆ computation and neighbor communication are both fully parallelized and their ratio remains constant in weak scaling
- **The programming model is SPMD/BSP/CSP**
 - ◆ Single Program, Multiple Data
 - ◆ Bulk Synchronous Programming
 - ◆ Communicating Sequential Processes
- **PETSc, Trilinos, hypre, etc.**

Estimating scalability

- **Given complexity estimates of the leading terms of:**
 - ◆ the concurrent computation (per iteration phase)
 - ◆ the concurrent communication
 - ◆ the synchronization frequency
- **And a model of the architecture including:**
 - ◆ internode communication (network topology and protocol reflecting horizontal memory structure)
 - ◆ on-node computation (effective performance parameters including vertical memory structure)
- **One can estimate optimal concurrency and optimal execution time**
 - ◆ on per-iteration basis
 - ◆ simply differentiate time estimate in terms of problem size N and processor number P with respect to P

3D stencil computation weak scaling

(assume fast local network, tree-based global reductions)

- Total wall-clock time per iteration (ignoring local comm.)

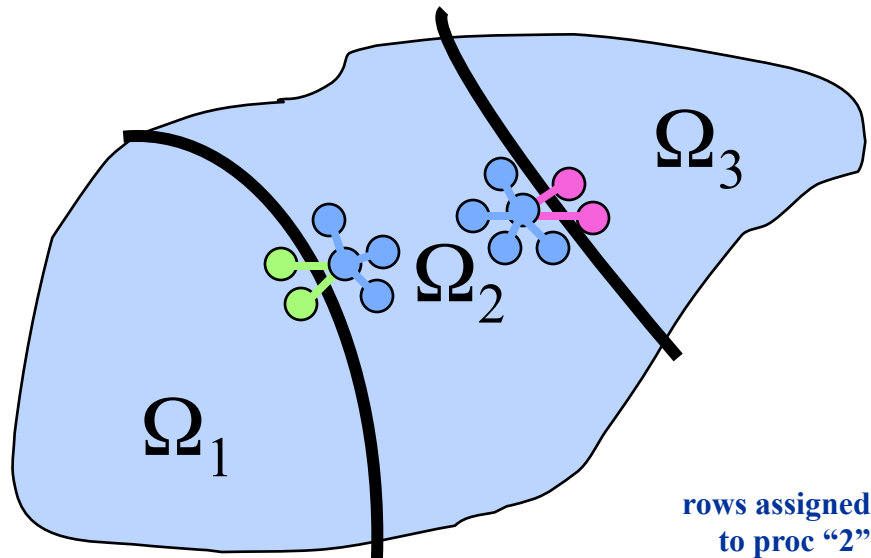
$$T(N, P) = A \frac{N}{P} + C \log P$$

- For optimal P , $\frac{\partial T}{\partial P} = 0$, or $-A \frac{N}{P^2} + \frac{C}{P} = 0$

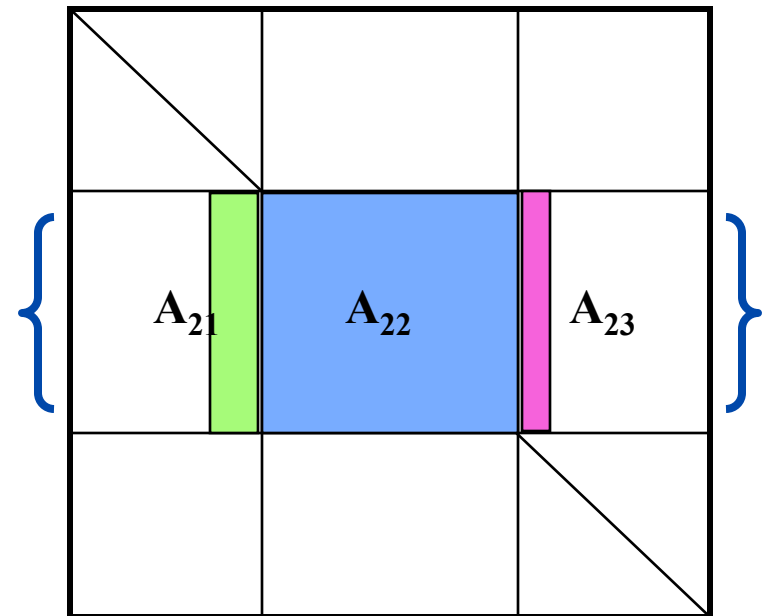
or $P_{opt} = \frac{A}{C} N$

- P can grow linearly with N , and running time increases “only” logarithmically – *as good as weak scaling can be!*
- Problems: (1) assumes perfect synchronization,
(2) log of a billion may be “large”

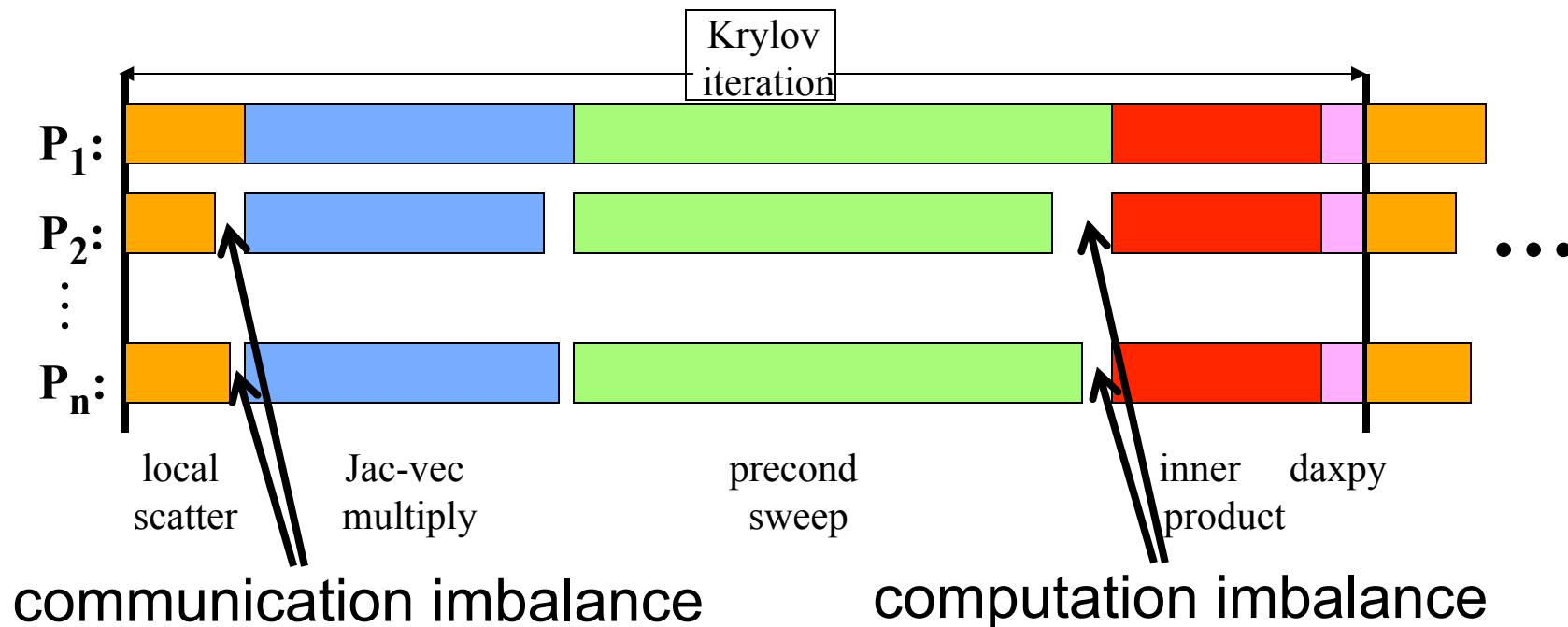
SPMD parallelism w/ domain decomposition: *an endangered species?*



**Partitioning of the grid
induces block structure on
the system matrix
(Jacobian)**

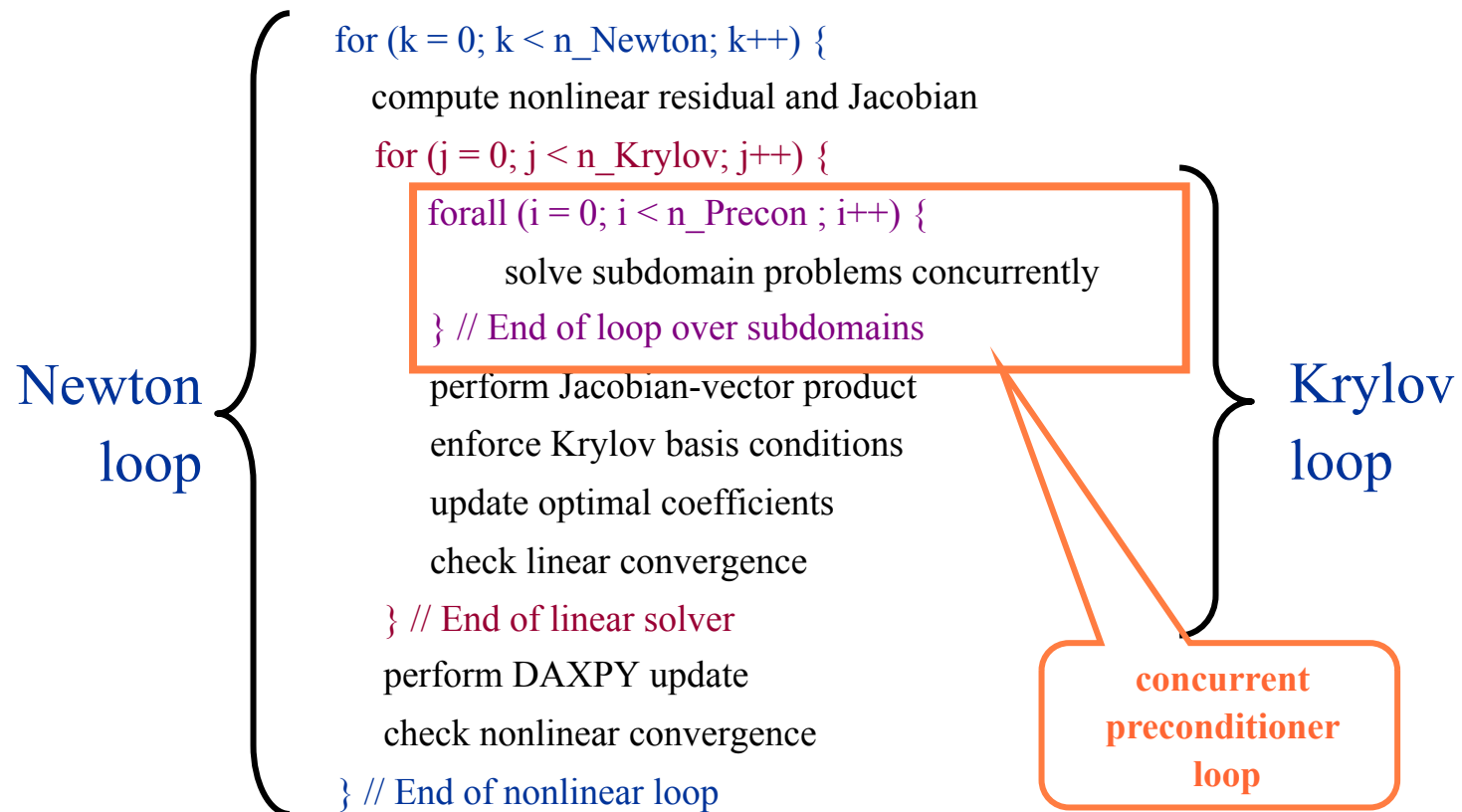


Workhorse innards: e.g., Krylov-Schwarz, a bulk synchronous implicit solver



Idle time due to load imbalance becomes a challenge at, say, one billion cores, when *one* processor can hold up *all* of the rest at a synchronization point

Our programming idiom is nested loops, e.g., Newton-Krylov-Schwarz

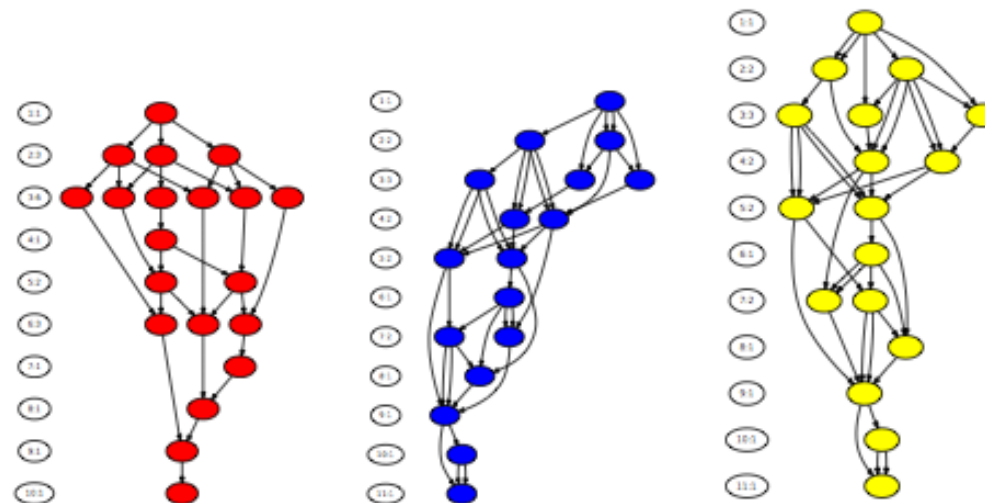


Outer loops (not shown): continuation, implicit timestepping, optimization

Dataflow illustration: generalized eigensolver

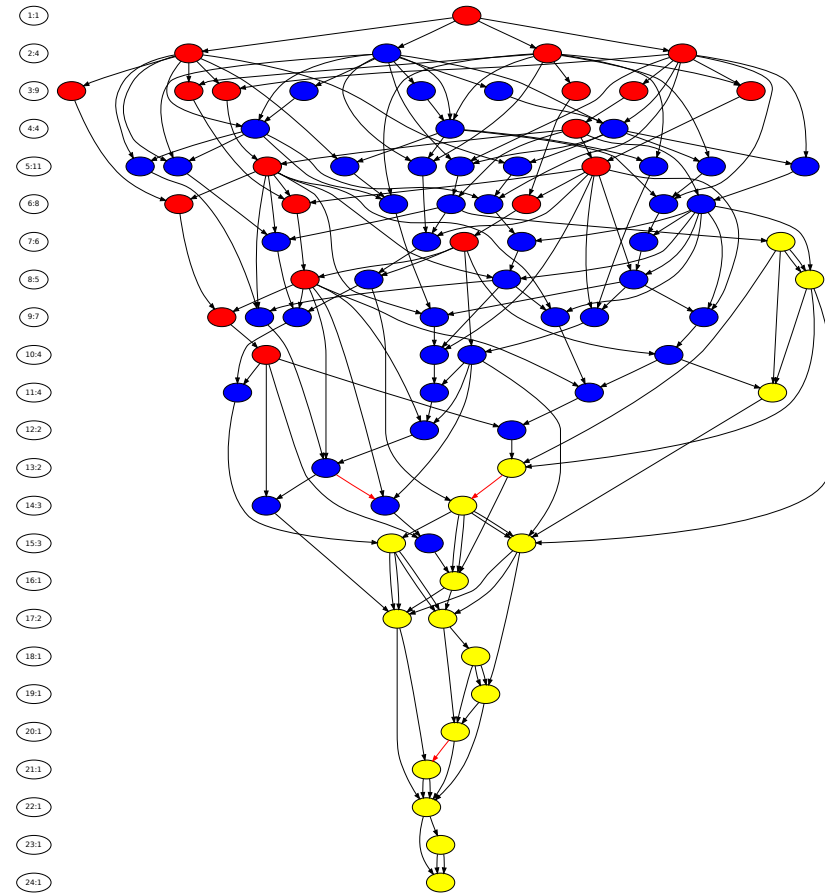
$$Ax = \lambda Bx$$

Operation	Explanation	LAPACK routine name
① $B = L \times L^T$	Cholesky factorization	POTRF
② $C = L^{-1} \times A \times L^{-T}$	application of triangular factors	SYGST or HEGST
③ $T = Q^T \times C \times Q$	tridiagonal reduction	SYEVD or HEEVD
④ $Tx = \lambda x$	QR iteration	STERF

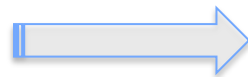
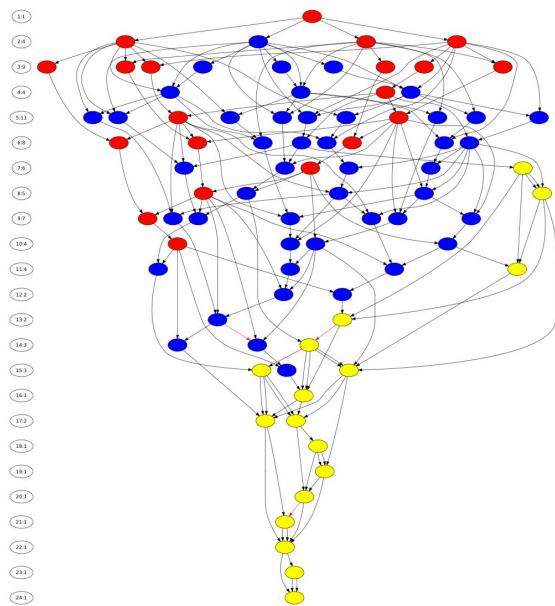


These loops, with their artifactual orderings, can be replaced with DAGs

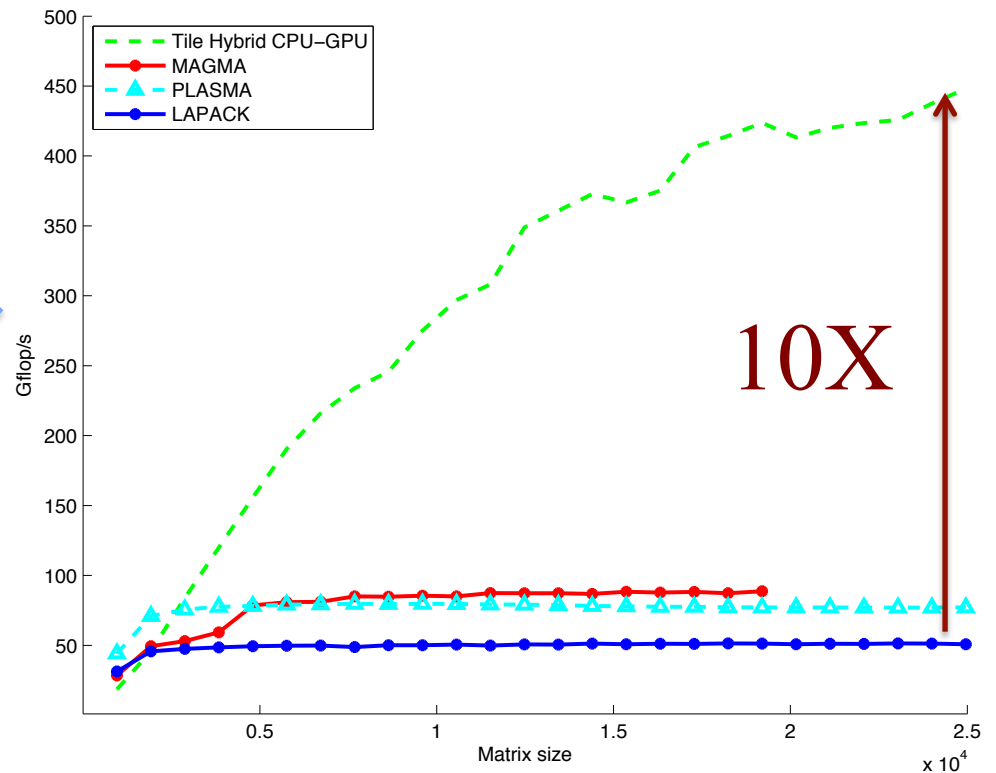
- Diagram shows a dataflow ordering of the steps of a 4×4 symmetric generalized eigensolver
- Nodes are tasks, color-coded by type, and edges are data dependencies
- Time is vertically downward



Co-variance matrix inversion on hybrid* CPU-GPU environment with DAG scheduling



Dynamic
Runtime
System

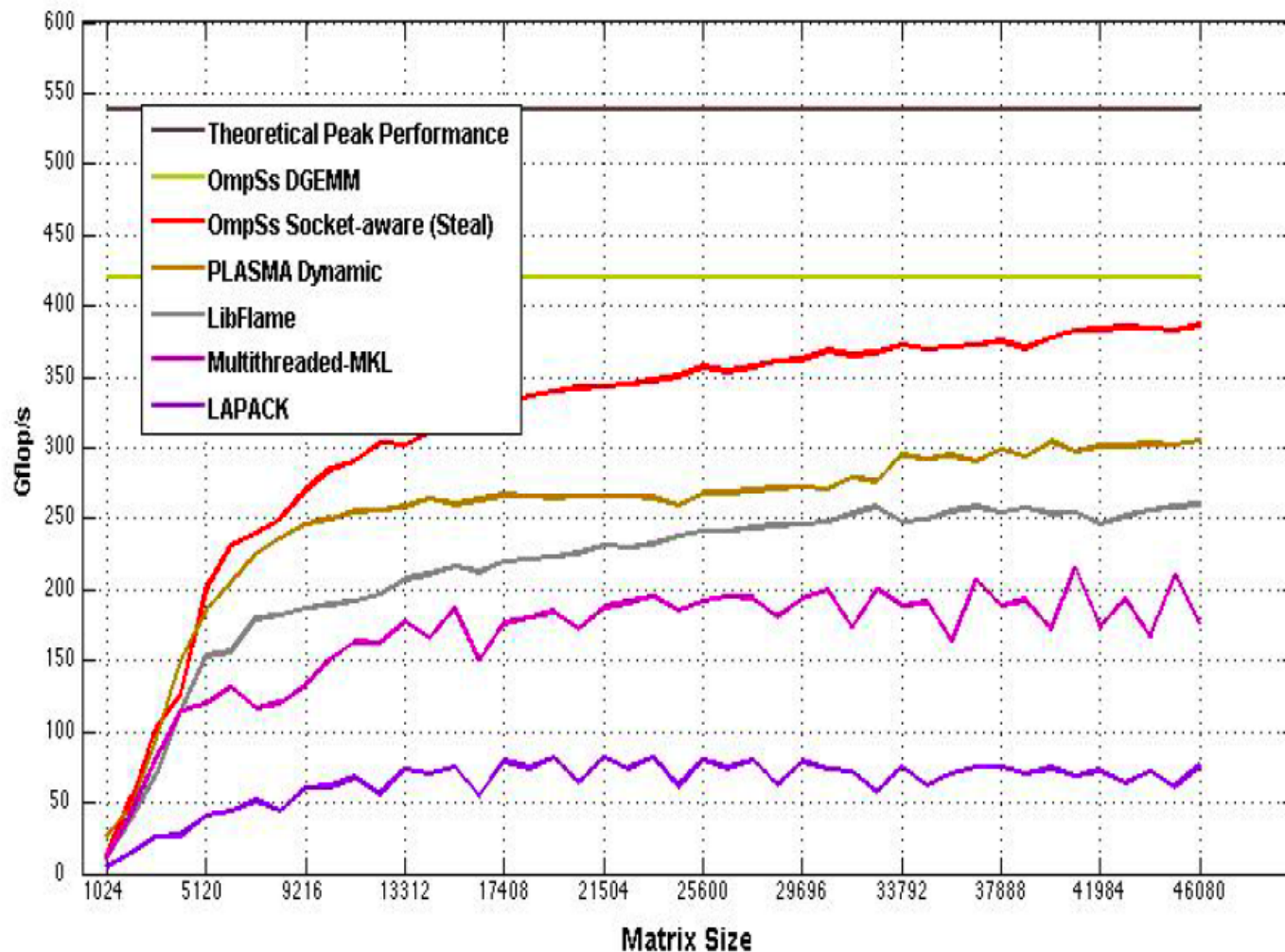


* On 8 Intel Xeons and 2 NVIDIA Teslas using StarPU (INRIA)

c/o Huda Ibeid (KAUST) *et al.* HiPC'11 (Bangalore)

ATPESC 2013

Research in progress: locality preserving work-stealing on Cholesky solver gets 93% of DGEMM*

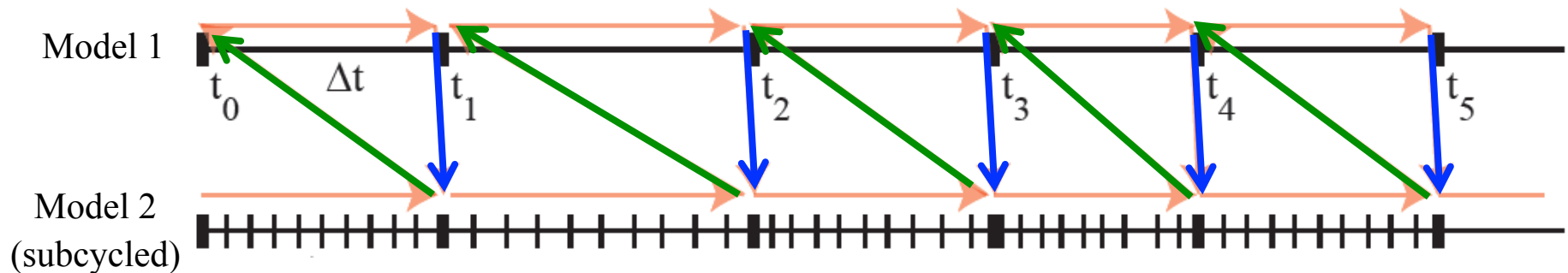


* On AMD “Istanbul” 8439 SE with 8 sockets, 6 cores per socket, using OmpSs

c/o Rabab AlOmairi (KAUST), MS thesis

ATPESC 2013

Multiphysics w/ legacy codes: *an endangered species?*



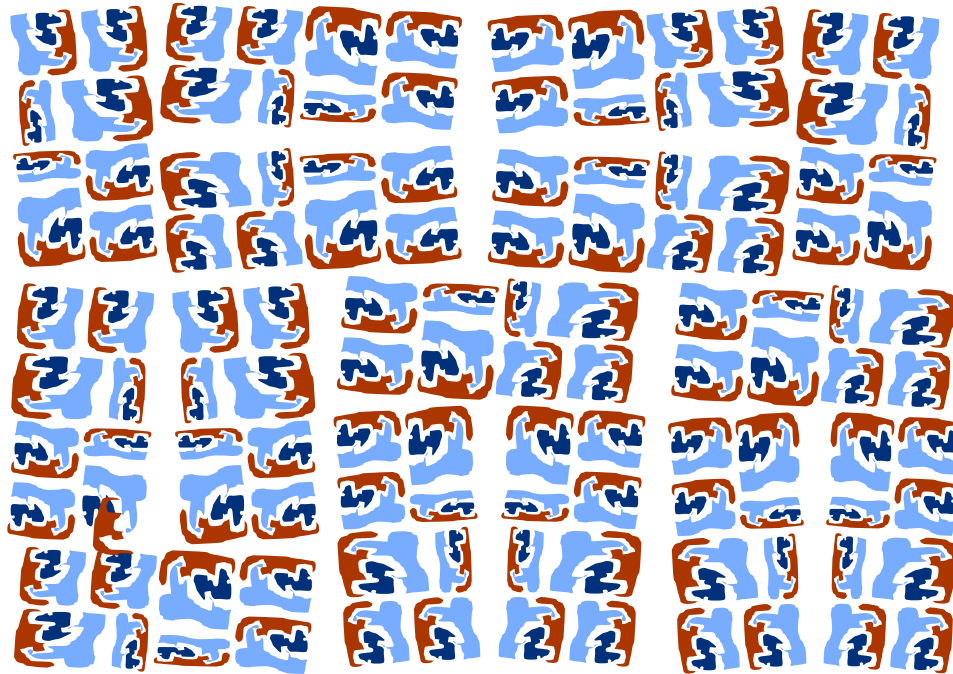
- Many multiphysics codes operate like this, where the models may occupy the same domain in the bulk (e.g., reactive transport) or communicate at interfaces (e.g., ocean-atmosphere)*
- The data transfer cost represented by the blue and green arrows may be much higher than the computation cost of the models, even apart from first-order operator splitting error and possible instability

*see "Multiphysics simulations: challenges and opportunities" (IJHPCA)

Many codes have the algebraic and software structure of multiphysics

- **Exascale is motivated by these:**
 - **uncertainty quantification, inverse problems, optimization, immersive visualization and steering**
- **These may carry auxiliary data structures to/from which blackbox model data is passed and they act like just another “physics” to the hardware**
 - **pdfs, Lagrange multipliers, etc.**
- **Today’s separately designed blackbox algorithms for these may not live well on exascale hardware: co-design may be required due to data motion**

Multiphysics layouts must invade blackboxes



- Each application must first be ported to extreme scale (distributed, hierarchical memory)
- Then applications may need to be interlaced at the data structure level to minimize copying and allow work stealing at synchronization points



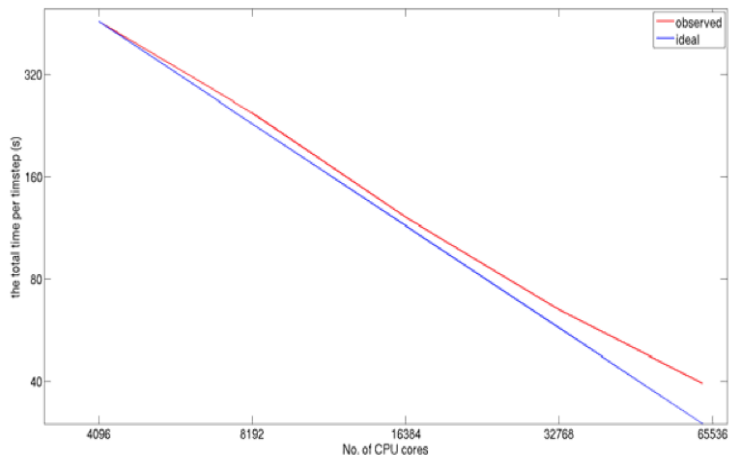
Multiphysics simulations: Challenges and opportunities

David E Keyes^{1,2}, Lois C McInnes³, Carol Woodward⁴,
William Gropp⁵, Eric Myra⁶, Michael Pernice⁷, John Bell⁸,
Jed Brown³, Alain Clo¹, Jeffrey Connors⁴, Emil Constantinescu³, Don Estep⁹,
Kate Evans¹⁰, Charbel Farhat¹¹, Ammar Hakim¹², Glenn Hammond¹³, Glen Hansen¹⁴,
Judith Hill¹⁰, Tobin Isaac¹⁵, Xiangmin Jiao¹⁶, Kirk Jordan¹⁷, Dinesh Kaushik³,
Efthimios Kaxiras¹⁸, Alice Koniges⁸, Kihwan Lee¹⁹, Aaron Lott⁴, Qiming Lu²⁰,
John Magerlein¹⁷, Reed Maxwell²¹, Michael McCourt²², Miriam Mehl²³,
Roger Pawlowski¹⁴, Amanda P Randles¹⁸, Daniel Reynolds²⁴, Beatrice Rivière²⁵,
Ulrich Rüde²⁶, Tim Scheibe¹³, John Shadid¹⁴, Brendan Sheehan⁹, Mark Shephard²⁷,
Andrew Siegel³, Barry Smith³, Xianzhu Tang²⁸, Cian Wilson² and Barbara Wohlmuth²³

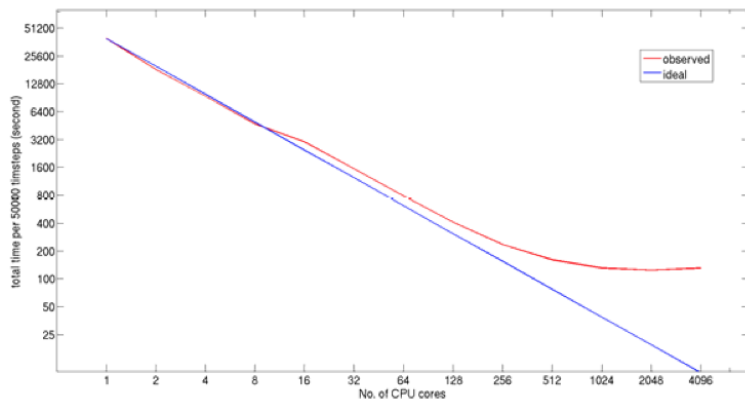
Abstract

We consider multiphysics applications from algorithmic and architectural perspectives, where "algorithmic" includes both mathematical analysis and computational complexity, and "architectural" includes both software and hardware environments. Many diverse multiphysics applications can be reduced, en route to their computational simulation, to a common algebraic coupling paradigm. Mathematical analysis of multiphysics coupling in this form is not always practical for realistic applications, but model problems representative of applications discussed herein can provide insight. A variety of software frameworks for multiphysics applications have been constructed and refined within disciplinary communities and executed on leading-edge computer systems. We examine several of these, expose some commonalities among them, and attempt to extrapolate best practices to future systems. From our study, we summarize challenges and forecast opportunities.

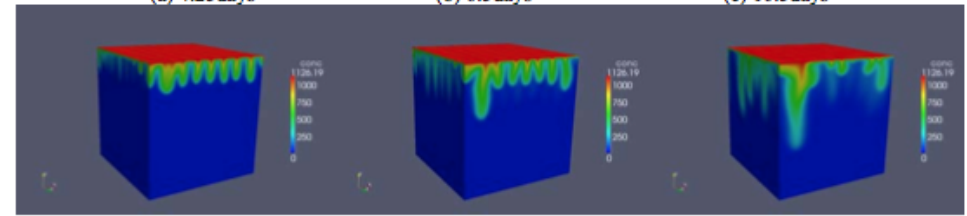
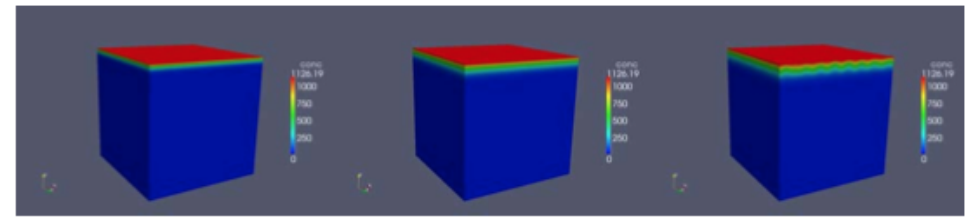
Multiphysics modeling of CO₂ sequestration by coupling PDEs and molecular dynamics



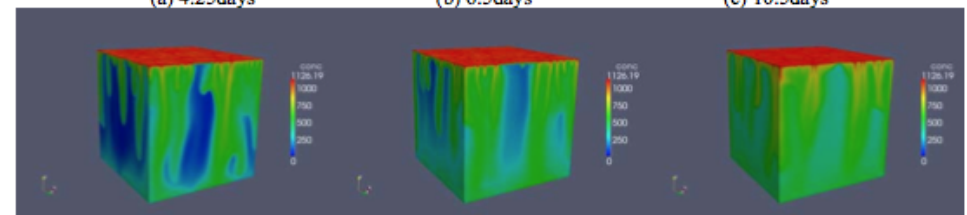
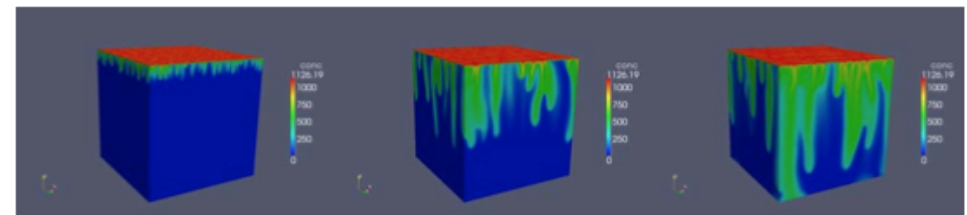
**Blue Gene/P strong scaling –
Reservoir Simulator**



**Blue Gene/P strong scaling –
Molecular Dynamics**



The concentration evolution over time with $0.6\Delta\rho_{sat}$



The concentration evolution over time with $2.0\Delta\rho_{sat}$

Bad news/good news (1)

- **One may have to explicitly control data motion**
 - carries the highest energy cost in the exascale computational environment
- **One finally will get the privilege of controlling the vertical data motion**
 - horizontal data motion under control of users under *Pax MPI*, already
 - but vertical replication into caches and registers was (until now with GPUs) scheduled and laid out by hardware and runtime systems, mostly invisibly to users

Bad news/good news (2)

- **“Optimal” formulations and algorithms may lead to poorly proportioned computations for exascale hardware resource balances**
 - **today’s “optimal” methods presume flops are expensive and memory and memory bandwidth are cheap**
- **Architecture may lure users into more arithmetically intensive formulations (e.g., fast multipole, lattice Boltzmann, rather than mainly PDEs)**
 - **tomorrow’s optimal methods will (by definition) evolve to conserve what is expensive**

Bad news/good news (3)

- **Hardware nonuniformity may force abandonment of the Bulk Synchronous Programming (BSP) paradigm**
 - **it will be impossible for the user to control load balance sufficiently to make it work**
- **Hardware and algorithmic nonuniformity will be indistinguishable at the performance level**
 - **good solutions for the dynamically load balancing in systems space will apply to user space, freeing users**

Bad news/good news (4)

- **Fully deterministic algorithms may simply come to be regarded as too synchronization-vulnerable**
 - **Rather than wait for data, we may infer it, taking into account sensitivity to poor guesses, and move on**
- **A rich numerical analysis of algorithms that make use of statistically inferred “missing” quantities may emerge**

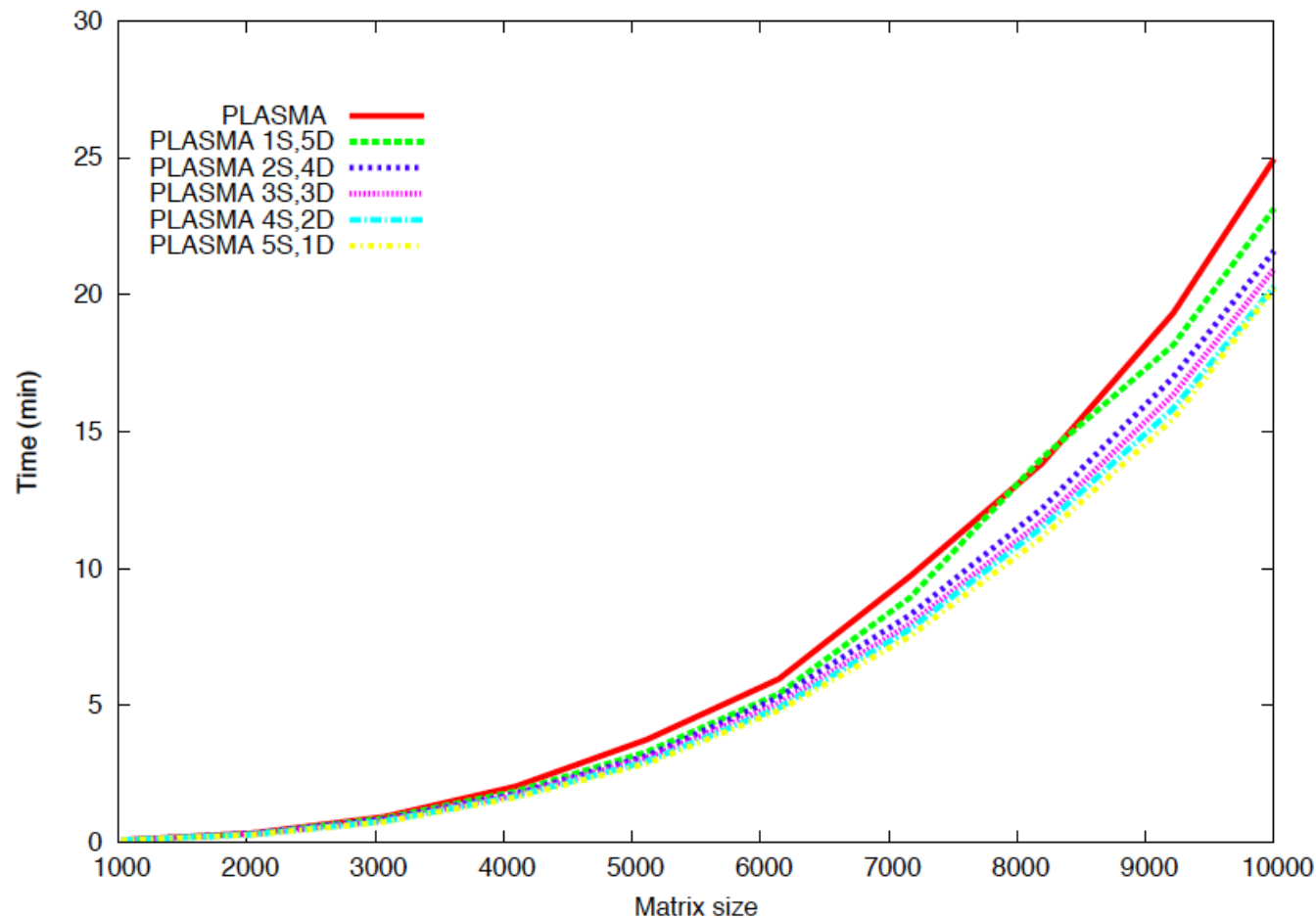
Bad news/good news (5)

- Fully reliable executions may simply come to be regarded as too costly/synchronization-vulnerable
- Algorithmic-based fault tolerance (ABFT) will be much cheaper than hardware and OS-mediated reliability
 - Developers will partition their data and their program units into two sets
 - A small set that must be done reliably (with today's standards for memory checking and IEEE ECC)
 - A large set that can be done fast and unreliably, knowing the errors can be either detected, or their effects rigorously bounded
- Examples in direct and iterative linear algebra
 - anticipated by Von Neumann, 1956 ("Synthesis of reliable organisms from unreliable components")

Bad news/good news (6)

- **Default use of (uniform) high precision may come to an end, as wasteful of storage and bandwidth**
 - **Representation of a smooth function in a hierarchical basis requires fewer bits than storing its nodal values**
 - **we will have to compute and communicate “deltas” between states rather than the full state quantities, as we did when double precision was expensive (e.g., iterative correction in linear algebra)**
 - **a combining network node will have to remember not just the last address, but also the last values, and send just the deltas**
- **Equidistributing errors properly while minimizing resource use will lead to innovative error analyses in numerical analysis**

Research in progress: reducing precision in the new QDHWeig eigensolver (Higham, 2013)



* Dual-socket 8-core (16 cores total), Intel(R) Xeon(R) CPU E5-2650

c/o Dalal Sukkari (KAUST), MS thesis

ATPESC 2013

How will PDE computations adapt?

- **Programming model will still be message-passing (due to large legacy code base), adapted to multicore or hybrid processors beneath a relaxed synchronization MPI-like interface**
- **Load-balanced blocks, scheduled today with nested loop structures will be separated into critical and non-critical parts**
- **Critical parts will be scheduled with directed acyclic graphs (DAGs)**
- **Noncritical parts will be made available for work-stealing in economically sized chunks**

Adaptation to asynchronous programming styles

- **To take full advantage of such asynchronous algorithms, we need to develop greater expressiveness in scientific programming**
 - ◆ **create separate threads for logically separate tasks, whose priority is a function of algorithmic state, not unlike the way a time-sharing OS works**
 - ◆ **join priority threads in a directed acyclic graph (DAG), a task graph showing the flow of input dependencies; fill idleness with noncritical work or steal work**
- **Steps in this direction**
 - ◆ **Asynchronous Dynamic Load Balancing (ADLB) [Lusk (Argonne), 2009]**
 - ◆ **Asynchronous Execution System [Steinmacher-Burrow (IBM), 2008]**

Evolution of Newton-Krylov-Schwarz: breaking the synchrony stronghold

- Can write code in styles that do not require artifactual synchronization
- Critical path of a nonlinear implicit PDE solve is essentially
... lin_solve, bound_step, update; lin_solve, bound_step, update ...
- However, we often insert into this path things that could be done less synchronously, because we have limited language expressiveness
 - ◆ Jacobian and preconditioner refresh
 - ◆ convergence testing
 - ◆ algorithmic parameter adaptation
 - ◆ I/O, compression
 - ◆ visualization, data mining

Sources of nonuniformity

- **System**

- ◆ **Already important: manufacturing, OS jitter, TLB/cache performance variations, network contention,**
- ◆ **Newly important: dynamic power management, more soft errors, more hard component failures, software-mediated resiliency, etc.**

- **Algorithmic**

- ◆ **physics at gridcell/particle scale (e.g., table lookup, equation of state, external forcing), discretization adaptivity, solver adaptivity, precision adaptivity, etc.**

- **Effects of both types are similar when it comes to waiting at synchronization points**

- **Possible solutions for system nonuniformity will improve programmability, too**

Programming practice

- **Prior to possessing exascale hardware, users can prepare themselves by exploring new programming models**
 - ◆ **on manycore and heterogeneous nodes**
- **Attention to locality and reuse is valuable at all scales**
 - ◆ **will produce performance paybacks today *and* in the future**
 - ◆ **domains of coherence will be variable and hierarchical**
- **New algorithms and data structures can be explored under the assumption that flop/s are cheap and moving data is expensive**

Path for scaling up applications

- “Weak scale” applications up to distributed memory limits
 - ◆ proportional to number of nodes
- “Strong scale” applications beyond this
 - ◆ proportional to cores per node/memory unit
- Scale the workflow, itself
 - ◆ proportional to the number of instances (ensembles)
 - ◆ integrated end-to-end simulation
- Algorithm-architecture co-design process is staged, with any of these types of scaling valuable by themselves
- Big question: does the software for co-design factor? Or is all the inefficiency at the data copies at interfaces between the components after a while?

Required software enabling technologies

Model-related

- ◆ Geometric modelers
- ◆ Meshers
- ◆ Discretizers
- ◆ Partitioners
- ◆ Solvers / integrators
- ◆ Adaptivity systems
- ◆ Random no. generators
- ◆ Subgridscale physics
- ◆ Uncertainty quantification
- ◆ Dynamic load balancing
- ◆ Graphs and combinatorial algs.
- ◆ Compression

Development-related

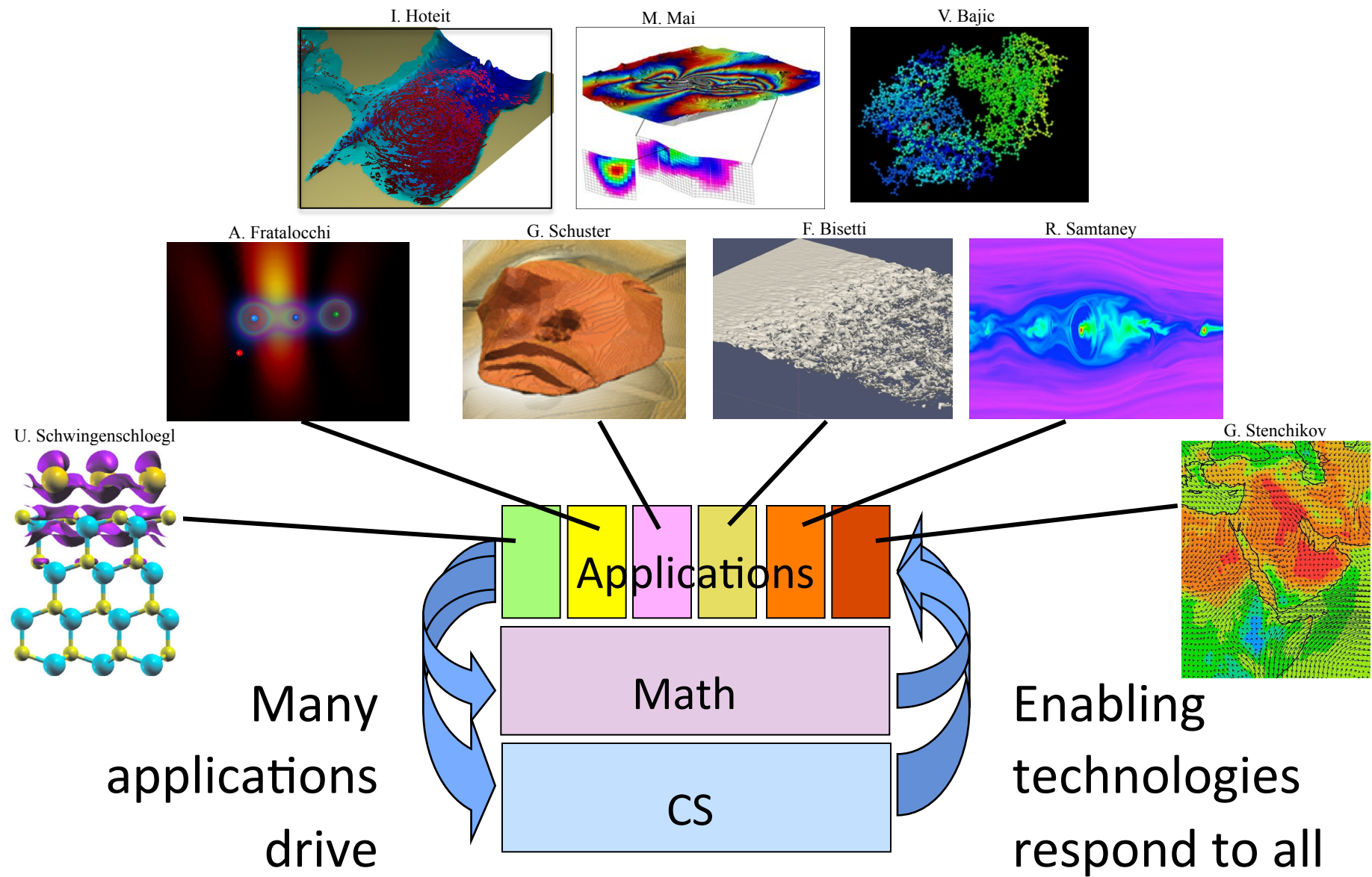
- ◆ Configuration systems
- ◆ Source-to-source translators
- ◆ Compilers
- ◆ Simulators
- ◆ Messaging systems
- ◆ Debuggers
- ◆ Profilers

High-end computers come with little of this stuff.
Most has to be contributed by the user community

Production-related

- ◆ Dynamic resource management
- ◆ Dynamic performance optimization
- ◆ Authenticators
- ◆ I/O systems
- ◆ Visualization systems
- ◆ Workflow controllers
- ◆ Frameworks
- ◆ Data miners
- ◆ Fault monitoring, reporting, and recovery

“SciDAC philosophy” of software investment

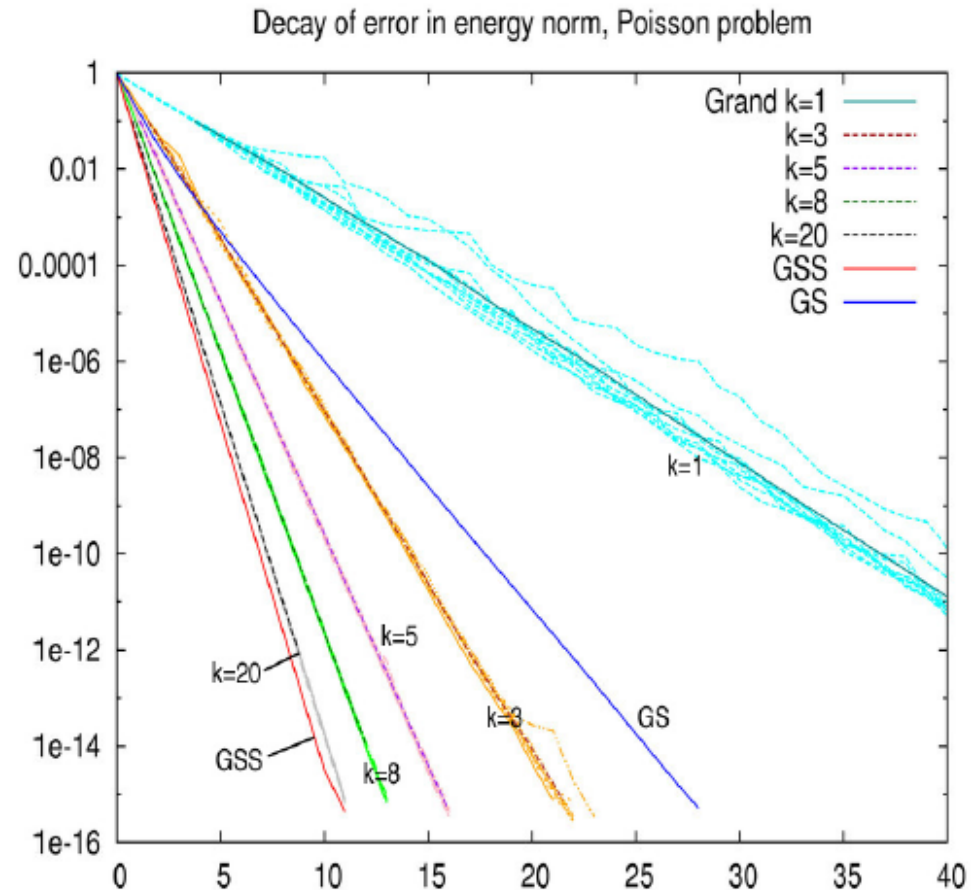


DOE's Exascale Mathematics Working Group

- **74 fascinating 2-page whitepapers contributed by the international community to the EMWG at**
<https://collab.mcs.anl.gov/display/examath/Submitted+Papers>
- **To be discussed this coming 20-21 August 2013 in DC**
 - **Randomized algorithms**
 - **On-the-fly data compression**
 - **Mining massive data sets**
 - **Algorithmic-based fault tolerance**
 - **Adaptive precision algorithms**
 - **Concurrency from dimensions beyond space (time, phase space, stochastic parameters)**
 - **etc.**

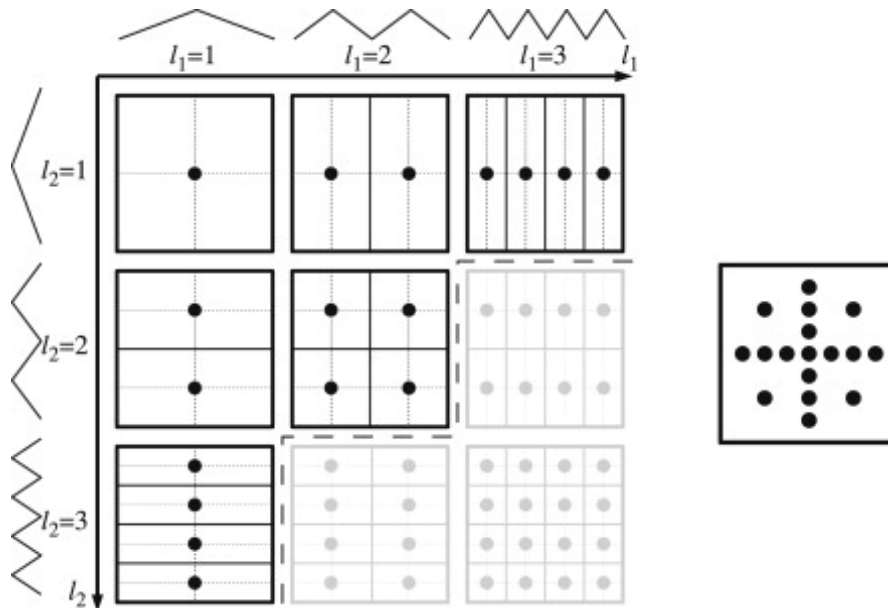
Randomized algorithms in subspace correction methods

- Solve $Ax=b$ by pointwise relaxation
- Gauss-Seidel (1823)
 - deterministic and pre-ordered
- Southwell (1935)
 - deterministic and dynamically ordered
- Griebel-Oswald (2011)
 - random (and dynamically ordered)
- Excellent convergence w/ fault tolerance and synchronization reduction



Hierarchical representations for extreme data

- Saving most simulation results to persistent storage will be impractical; instead hybrid *in situ* / *in transit* analysis
- Challenges:
 - On-the-fly compression
- Algorithmic idea: sparse grids



Storage complexity

$$O(n^d) \rightarrow O(n \cdot (\log n)^{d-1})$$

(spatial dimension d)

Representation accuracy

$$O(n^{-p}) \rightarrow O(n^{-p} \cdot (\log n)^k)$$

(order p ; k depends on p , d)

How do sparse representations work?

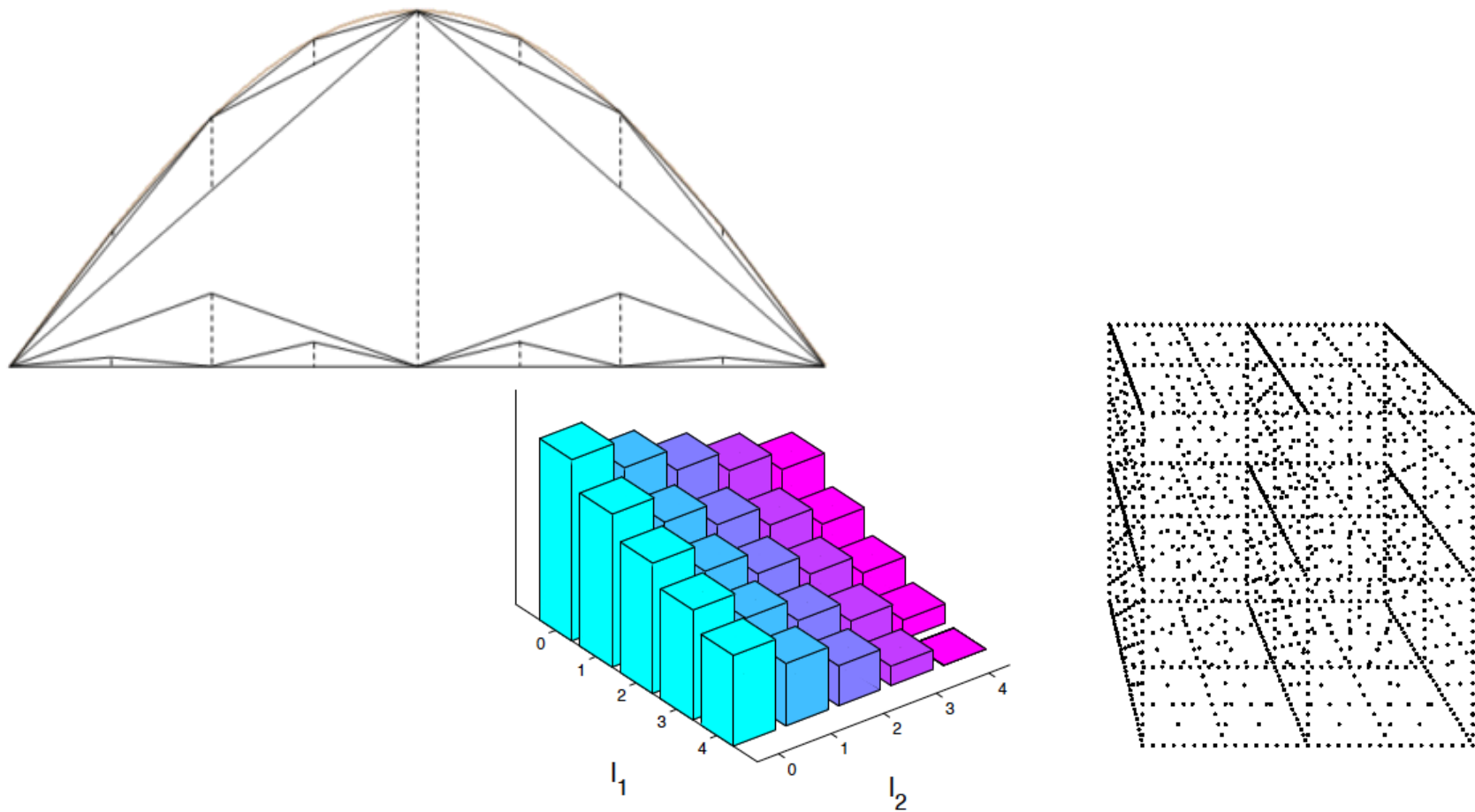


Fig. 1. Left: Norms of errors of the difference spaces W_1 on a logarithmic scale. The example function is $u(x_1, x_2) = e^{-(x_1^2 + x_2^2)}$ with $(x_1, x_2) \in [0, 1] \times [0, 1]$. Right: Sparse grid of refinement level $l = 5$ in three dimensions

Acknowledgment: today's Peta-op/s machines



10^{12} neurons @ 1 KHz = 1 PetaOp/s
1.4 kilograms, 20 Watts

See 2011 special issue of *Comptes Rendus*



Exaflop/s: The why and the how, D. E. Keyes, *Comptes Rendus de l'Académie des Sciences* **339**, 2011, 70—77.

Thank you



KAUST is
recruiting! Your
office here 😊

شكرا

david.keyes@kaust.edu.sa